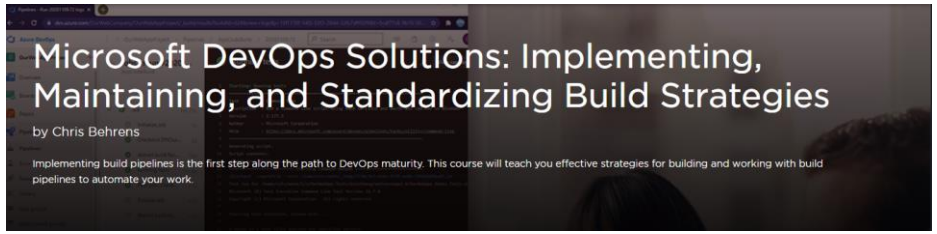
Microsoft DevOps Solutions: Designing Build Automation

MAKING YOUR BUILD WORK FOR YOU

Chris B. Behrens
SOFTWARE ARCHITECT
@chrisbbehrens

A (Very) Fast-paced Course



https://app.pluralsight.com/library/courses/microsoft-devops-solutions-implementing-maintaining-standardizing-build-strategies

https://app.pluralsight.com/library/courses/microsoft-azure-creating-automated-build-workflow

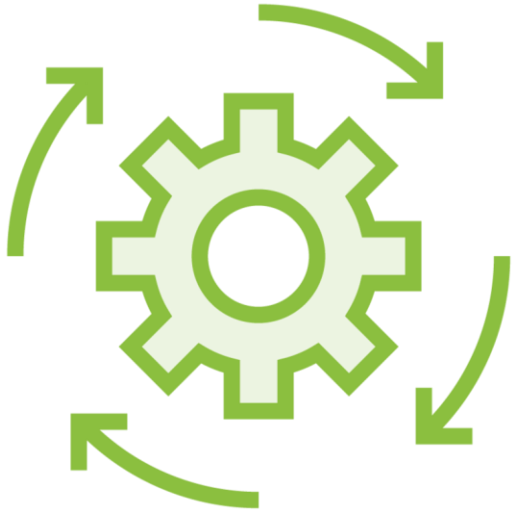
https://app.pluralsight.com/library/courses/microsoft-azure-monitoring-code-quality

https://app.pluralsight.com/library/courses/running-jenkins-docker

https://app.pluralsight.com/library/courses/automating-jenkins-groovy

The Purpose of a Build

Your build is a fuse which blows up before your customer does



Compilation is the first milestone

Maturity level one

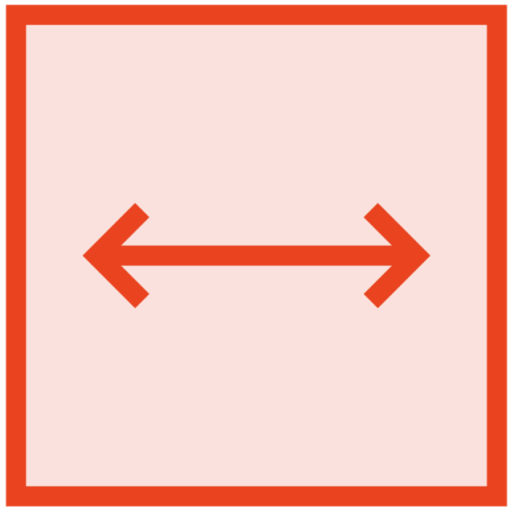
What Comes Next

It compiles, but is it right?

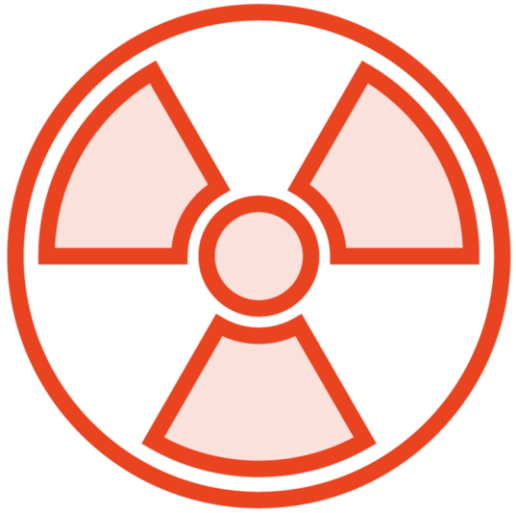
That is, does it meet expectations?

Untested code has an unknown state

Untested Code



We don’t test dependencies



We test code we care about

Ultimately, we should care about everything

What Comes after Test Coverage

We've got useful coverage

Correctness is demonstrated

What else can we measure?

Some Code with Problems

```
var connection = new DatabaseConnection();


connection.UserName = "admin";

connection.Password = "Password123";


connection.Open();
```

1. You can’t change them without re-deploying the code

2. You can’t have environment-specific values

3. The secret is in version control

Secrets do not belong in version control

Some Code with Problems...?

```
var superEngine = new Engine();


superEngine.ClassName = "admin";

superEngine.TestEngineTitle = "Password123";


superEngine.Open();
```

Important Build Tools

There are a lot of build tools

Examination on content

And on performance

Static = content

Dynamic = performance

The exam covers specific tools

Static Analysis

Via rules which human beings have figured out

Implemented via patterns which can be scanned for

When found, the code is flagged

And connected with information for remediation

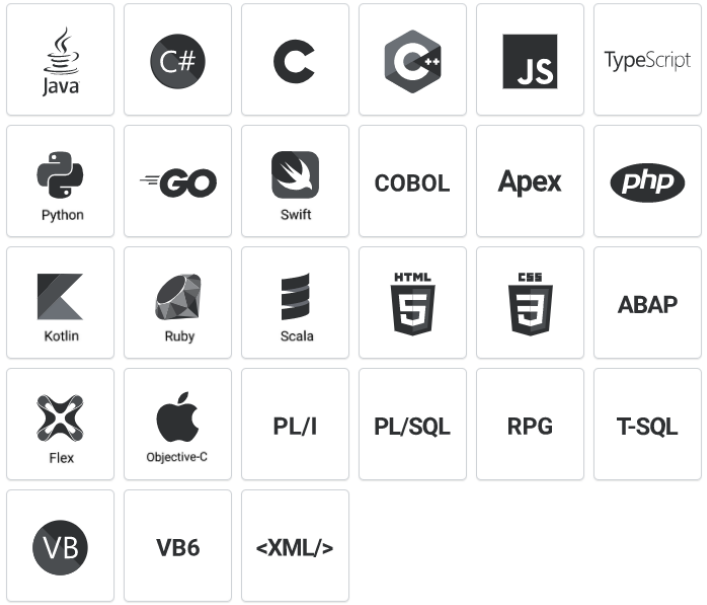
A suite of tools working in concert

A Roslyn analyzer

A build scanner

An online portal

Setting up SonarQube with Your Build



1. Within SonarCloud, create an API token
2. Within Azure DevOps install the Marketplace plug-in
3. Create a service connection to SonarQube and provide the API token

PMD

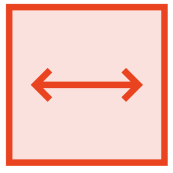
Programming Mistake Detector

Copy & paste detection

SonarQube can execute PMD tasks

Why choose? All of the above

Open Source Licensing and Vulnerabilities



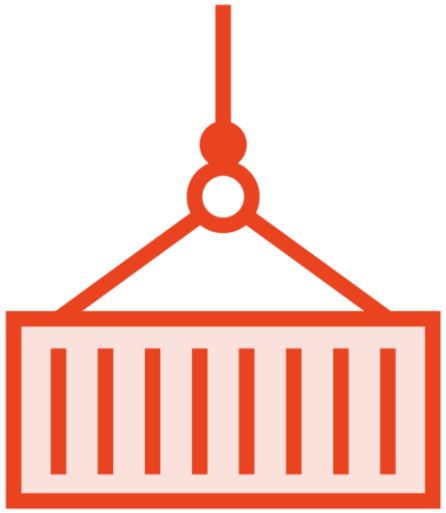
Dependency management

All modern applications leverage third-party dependencies

If you're working in the Microsoft stack, you rely on .Net

If you're writing Java code, then you rely on the Java runtime

Open Source Packaging



These days, dependencies are packages

With Microsoft, it's usually a Nuget package

Consider who you trust, how much, and why

A Jenkins Plug-in Story

Running Jenkins in Docker

A build engine, like Azure Pipelines

Jenkins relies *heavily* on third-party plug-ins

I was using the Docker plug-in

- A developer (of another plug-in) broke it
- With a bad code push

I noticed it immediately because I was working with it heavily

But it might have gone unnoticed for a long time

An automated process to find the problem would have been great

A Licensing Story

Automated transformation of Word docs to HTML

Word documents are zip files

I used an open source compression library

The GNU License

Coded, tested, delivered, supported

The GNU license

My application had to be free, too

The MIT license would have been better

How About You?

A tool to check these problems

Identify and clarify

And maybe just buy something, instead

Software Composition Analysis

- WhiteSource Bolt
- BlackDuck
- Analyzing the composition of your software
- Analysis and report within the build
- Snyk
- Snyk can also scan Docker containers

Dynamic Security Analysis

SonarQube will find many security problems

WhiteSource Bolt can find package vulnerabilities

Analyzing *behavior* is Dynamic Analysis

How Dynamic Analysis Works

1. The code must be compiled
2. The code must be deployed*

"Deployment" can mean a lot of things

Compiling the code and executing it inside a test harness

OWASP

OWASP – the Open Web Application Security Project

Zed Attack Proxy

OWASP Top Ten Security Vulnerabilities

Zed Attack Proxy

A comprehensive catalog of attacks

You can use ZAP in your build

Then ZAP actually executes those attacks against your code

Tool Summary

1. **SonarQube – static analysis that finds all kinds of problems in your code**
2. **Software Composition Analysis Tools:**
   - a. WhiteSource Bolt
   - b. Black Duck (and)
   - c. Snyk
3. **Zed Attack Proxy – a dynamic analysis tool which executes lightweight security penetration tests against your deployed code**

What a Quality Gate Is and Why You Need It

A quality gate breaks the build

Build gates are not the only kind of quality gate

Commit Quality Gates

Developers forget to associate commits with work items

Unless there's a rule that makes them

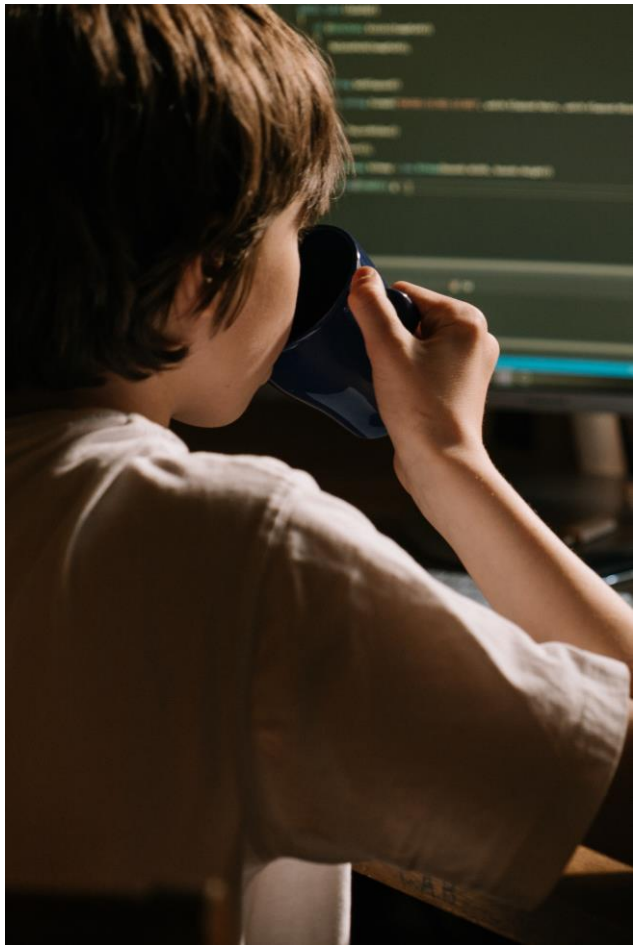
1. Traceability
2. Focus

The Quality Gate of Unit Testing

The execution of unit tests is also a quality gate

When a unit test breaks your build, you are protected

Meaningful Tests



The new developer writes bad tests

Bad testing creates a vicious cycle

Defects accumulate…

Which slows down development

Which makes schedules tighter

Which makes you further skimp on quality

To have a *chance* of escaping this…

- You must have coverage
- And not in a trivial way

Only human beings can determine meaning

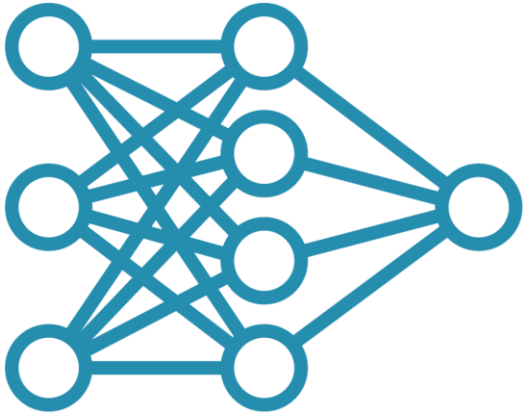
Because only human beings can determine intention

Testing Meaningful Code

Let's stipulate that we've got meaningful tests

Are you testing the meaningful part of the code?

Or are you testing what's easily testable?



That tends to fall by the wayside

Alright...

- Our tests are meaningful
- And we're testing meaningful code

We can enforce that with coverage

Let your coverage percentage pull the requirement

Rather than trying to push it

Otherwise, your build will fail as is

And you'll end up bypassing that check